Host Services Function Descriptions (version 4.1a)

The Host Services provide an interface between a host computer and a DSK to permit the development of stand-alone applications that control the DSK,

Supported hardware includes the TMS320C6211, TMS320C6711, TMS320C6713, and TMS320C6416T DSKs. To use the TMS320C6713 or TMS320C6416T DSKs, the DSK6XXXHPI daughtercard must be installed on the board. The TMS320C6416T DSK requires daughtercard software version 1.0.1.0 or higher. You can use winDSK6 version 4.0.1.0 or higher to verify the daughtercard software version.

The *StartUp* function must be called first. The *setDSP*, *SetLptPort*, and *SetComPort* functions must then be called prior to any other functions. The *ResetAndRunCOFF* or *ResetAndOpenHPI* function must be called prior to calling the *Read*, *Write*, *ReadFloat*, or *WriteFloat* functions. The example applications demonstrate the proper function usage.

The functions that make up the Host Services interface are:

- bool C6XCONTROL_StartUp();
- void C6XCONTROL_setDSP(DSP_VERSION version);
- void C6XCONTROL_setLptPort(PrinterPortNumbers port, PrinterPortModes mode);
- void C6XCONTROL_setComPort(ComPortNumbers port, ComSpeeds speed);
- int C6XCONTROL_ResetDsk();
- bool C6XCONTROL_ResetAndRunCOFF(const char *filename, bool getHPI_Link);
- unsigned long C6XCONTROL_getHPI_BaseAddress();
- bool C6XCONTROL_ResetAndOpenHPI();
- bool C6XCONTROL_Run();
- bool C6XCONTROL_Write(unsigned long address, unsigned long count, unsigned long* buffer);
- bool C6XCONTROL_Read(unsigned long address, unsigned long count, unsigned long* buffer);
- bool C6XCONTROL_WriteFloat(unsigned long address, unsigned long count, float* buffer);
- bool C6XCONTROL_ReadFloat(unsigned long address, unsigned long count, float* buffer);
- unsigned long C6XCONTROL_getVersion();
- bool C6XCONTROL_GetSymbolValue(const char *symbol, unsigned long* value);

Individual functions are described in detail on the following pages.

C6XCONTROL_StartUp

bool C6XCONTROL_StartUp();

Return Value

Returns false if the operating system is Windows NT/2000/XP and the parallel port drivers cannot be loaded, and returns true otherwise. In Windows NT/2000/XP, the drivers are required in order to perform parallel port input/output. If the parallel port interface is NOT going to be used, this return value can be ignored. If the parallel port interface is used, the program will fail if this function did not return true.

Parameters

None.

Remarks

Performs the basic initialization of the DSK software interface, including loading the parallel port driver if operating under Windows NT/2000/XP.

This function MUST be called before calling any other functions.

C6XCONTROL_setDSP

void C6XCONTROL_setDSP(DSP_VERSION version);

Return Value

None.

Parameters

version

Indicates the DSK that will be interfaced with. This parameter must one of the values enumerated in *windsk6_enums.h*.

C6211 - TMS320C6211 DSK

C6711 - TMS320C6711 DSK

C6713_LPT - TMS320C6713 DSK when using the parallel port interface

C6713_COM - TMS320C6713 DSK when using the serial port or USB interface

C6416T_LPT - TMS320C6416T DSK when using the parallel port interface

C6416T_COM- TMS320C6416T DSK when using the serial port or USB interface

Remarks

This function must be called after ***StartUp***.

C6XCONTROL_setLptPort

void C6XCONTROL_setLptPort(PrinterPortNumbers port, PrinterPortModes mode);

Return Value

None.

Parameters

port

Sets the parallel port number that the DSK will be interfaced with. This parameter must one of the values enumerated in *windsk6_enums.h*.

LPT1 - Parallel port with base address 0x0378

LPT2 - Parallel port with base address 0x0278

LPT3 - Parallel port with base address 0x03BC

mode

Sets the parallel port mode for the port that the DSK will be interfaced with. This parameter must one of the values enumerated in *windsk6_enums.h*.

SPP_NIBBLE_MODE - Sets port to SPP mode, using nibble mode for reads. Must be used on ports that do not support bidirectional operation.

SPP_MODE - Sets port to bidirectional SPP mode

EPP_1_7_MODE - Sets port to EPP 1.7 mode

EPP_MODE - Sets port to standard EPP mode

EPP16_MODE - Sets port to EPP mode with 16-bit transfers

Remarks

This function is used if the DSK will be interfaced with using the parallel port. The TMS320C6211 and TMS320C6711 DSKs are only supported using the parallel port interface. The parallel port interface on the DSK6XXXHPI daughtercard does not support EPP modes.

C6XCONTROL_setComPort

void C6XCONTROL_ setComPort(ComPortNumbers port, ComSpeeds speed);

Return Value

None.

Parameters

port

Sets the serial port number that the DSK will be interfaced with. This parameter must one of the values enumerated in *windsk6_enums.h*.

COM1, COM2, COM3, COM4, COM5, COM6, COM7, COM8

speed

Sets the serial port mode for the port that the DSK will be interfaced with. This parameter must one of the values enumerated in *windsk6_enums.h*.

COM_RS232 - Used for communications using the serial port on the TMS320C6713 HPI daughtercard.

COM_USB2 - Used for communications using the USB port on the TMS320C6713 HPI daughtercard.

Remarks

This function is used if the DSK6XXXHPI daughtercard will be interfaced with using the serial or USB port. The USB interface is implemented as a virtual COM port using the CP-2102 drivers.

C6XCONTROL_ResetDsk

bool C6XCONTROL_ResetDsk();

Return Value

Returns true on success. A false return values indicates a problem on the DSK or an error communicating with the DSK.

Parameters

None.

Remarks

Performs a reset of the DSK, then loads and executes a DSK program that disables all interrupts.

C6XCONTROL_ResetAndRunCOFF

bool C6XCONTROL_ResetAndRunCOFF(const char *filename, bool getHPI_Link);

Return Value

Returns true on success. A false return values indicates a problem on the DSK or an error communicating with the DSK.

Parameters

filename

Pointer a string containing the filename to load to the DSK.

getHPI_Link

Set to true to attempt to load the address of the HPI_Block variable, if you are using an HPI_Block variable in your CCS program as shown in the example. Only set to false if you are not using an HPI_Block variable. If set to false, the function C6XCONTROL_ getHPI_BaseAddress should NOT be called.

Remarks

Performs a reset of the DSK, then loads and executes the program specified in *filename* to the DSK.

C6XCONTROL_ getHPI_BaseAddress

bool C6XCONTROL_ getHPI_BaseAddress();

Return Value

Returns true on success. A false return values indicates a problem on the DSK or an error communicating with the DSK.

Parameters

None.

Remarks

Returns the base address of the HPI_Block structure on the DSK, as stored in a designated location in the DSK's memory space. The HPI block is used in the sample software application to permit easy identification of the address where variables are stored in memory.

C6XCONTROL_ResetAndOpenHPI

bool C6XCONTROL_ResetAndOpenHPI();

Return Value

Returns true on success. A false return values indicates a problem on the DSK or an error communicating with the DSK.

Parameters

None.

Remarks

Resets the DSK, opens the HPI connection, and initializes the DSK (EMIF and PLL, if applicable).

This function can be used if the user wants to load a program in other than COFF format. First, call this function, then load the program using the C6XCONTROL_Write function. When the program is loaded, call the C6XCONTROL_Run to start DSP execution.

C6XCONTROL_Run

bool C6XCONTROL_Run();

Return Value

Returns true on success. A false return values indicates a problem on the DSK or an error communicating with the DSK.

Parameters

None.

Remarks

Generates an HPI interrupt to start DSP execution after reset. This function should only be used in conjunction with the C6XCONTROL_ResetAndOpenHPI function.

C6XCONTROL_ Write
C6XCONTROL_ Read

bool C6XCONTROL_Write(unsigned long address, unsigned long count, unsigned long* buffer);
bool C6XCONTROL_Read(unsigned long address, unsigned long count, unsigned long* buffer);

Return Value

Returns true on success. A false return values indicates a problem on the DSK or an error communicating with the DSK.

Parameters

address

The address in the DSK's memory space to transfer to or from.

count

The number of words (32-bit) to transfer.

buffer

A pointer to the data buffer on the host for the transfer.

Remarks

Writes or reads the number of 32-bit words specified, starting at the specified address in the DSK's memory space and using the specified buffer on the host.

It is the caller's responsibility to ensure that the buffer is of adequate size, and that transfers at the given address on the DSK will not cause problems to the DSK's operation.

The host port interface on the supported DSKs only permits transfers of 32-bit data to be made.

C6XCONTROL_ WriteFloat
C6XCONTROL_ ReadFloat

bool C6XCONTROL_WriteFloat (unsigned long address, unsigned long count, float* buffer);
bool C6XCONTROL_ReadFloat (unsigned long address, unsigned long count, float* buffer);

Return Value

Returns true on success. A false return values indicates a problem on the DSK or an error communicating with the DSK.

Parameters

address

The address in the DSK's memory space to transfer to or from.

count

The number of floats (32-bit) to transfer.

buffer

A pointer to the data buffer on the host for the transfer.

Remarks

Writes or reads the number of 32-bit floats specified, starting at the specified address in the DSK's memory space and using the specified buffer on the host.

It is the caller's responsibility to ensure that the buffer is of adequate size, and that transfers at the given address on the DSK will not cause problems to the DSK's operation.

The host port interface on the supported DSKs only permits transfers of 32-bit data to be made.

C6XCONTROL_getVersion

unsigned long C6XCONTROL_getVersion();

Return Value

Returns the version number of the winDSK6 kernel software the DLL was built with.

Parameters

None.

Remarks

The version is encoded as four unsigned 8-bit numbers.

C6XCONTROL_GetSymbolValue

bool C6XCONTROL_GetSymbolValue(const char *symbol, unsigned long* value);

Return Value

Returns true if the symbol was located, and false otherwise

Parameters

symbol

The symbol name to look up. Identifiers in C will need to have a leading underscore appended, per standard C naming convention .

value

Pointer to a variable to store the symbol's value in.

Remarks

The program's symbol table is searched for the symbol name passed in. If it is located, the value is returned indirectly in the caller's variable. If the symbol does not exist in the program file, the function returns false.